NAG Library Function Document

nag_dgesvx (f07abc)

1 Purpose

nag_dgesvx (f07abc) uses the LU factorization to compute the solution to a real system of linear equations

$$AX = B \quad \text{or} \quad A^{\mathrm{T}}X = B,$$

where A is an n by n matrix and X and B are n by r matrices. Error bounds on the solution and a condition estimate are also provided.

2 Specification

```
#include <nag.h>
#include <nagf07.h>
```

```
void nag_dgesvx (Nag_OrderType order, Nag_FactoredFormType fact,
     Nag_TransType trans, Integer n, Integer nrhs, double a[], Integer pda,
     double af[], Integer pdaf, Integer ipiv[], Nag_EquilibrationType *equed,
     double r[], double c[], double b[], Integer pdb, double x[],
     Integer pdx, double *rcond, double ferr[], double berr[],
     double *recip_growth_factor, NagError *fail)
```

3 Description

nag_dgesvx (f07abc) performs the following steps:

1. **Equilibration**

   The linear system to be solved may be badly scaled. However, the system can be equilibrated as a first stage by setting **fact** = Nag_EquilibrateAndFactor. In this case, real scaling factors are computed and these factors then determine whether the system is to be equilibrated. Equilibrated forms of the systems $AX = B$ and $A^{\mathrm{T}}X = B$ are

$$(D_RAD_C)\left(D_C^{-1}X\right) = D_RB$$

   and

$$(D_RAD_C)^{\mathrm{T}}\left(D_R^{-1}X\right) = D_CB,$$

   respectively, where D_R and D_C are diagonal matrices, with positive diagonal elements, formed from the computed scaling factors.

   When equilibration is used, A will be overwritten by D_RAD_C and B will be overwritten by D_RB (or D_CB when the solution of $A^{\mathrm{T}}X = B$ is sought).

2. **Factorization**

   The matrix A, or its scaled form, is copied and factored using the LU decomposition

$$A = PLU,$$

   where P is a permutation matrix, L is a unit lower triangular matrix, and U is upper triangular.

   This stage can be by-passed when a factored matrix (with scaled matrices and scaling factors) are supplied; for example, as provided by a previous call to nag_dgesvx (f07abc) with the same matrix A.

3. **Condition Number Estimation**

   The LU factorization of A determines whether a solution to the linear system exists. If some diagonal element of U is zero, then U is exactly singular, no solution exists and the function returns with a failure. Otherwise the factorized form of A is used to estimate the condition number of the matrix A. If the reciprocal of the condition number is less than ***machine precision*** then a warning code is returned on final exit.

4. **Solution**

   The (equilibrated) system is solved for X ($D_C^{-1}X$ or $D_R^{-1}X$) using the factored form of A ($D_R A D_C$).

5. **Iterative Refinement**

   Iterative refinement is applied to improve the computed solution matrix and to calculate error bounds and backward error estimates for the computed solution.

6. **Construct Solution Matrix *X***

   If equilibration was used, the matrix X is premultiplied by D_C (if **trans** = Nag_NoTrans) or D_R (if **trans** = Nag_Trans or Nag_ConjTrans) so that it solves the original system before equilibration.

4 References

Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J J, Du Croz J J, Greenbaum A, Hammarling S, McKenney A and Sorensen D (1999) *LAPACK Users' Guide* (3rd Edition) SIAM, Philadelphia http://www.netlib.org/lapack/lug

Golub G H and Van Loan C F (1996) *Matrix Computations* (3rd Edition) Johns Hopkins University Press, Baltimore

Higham N J (2002) *Accuracy and Stability of Numerical Algorithms* (2nd Edition) SIAM, Philadelphia

5 Arguments

1: **order** – Nag_OrderType *Input*

On entry: the **order** argument specifies the two-dimensional storage scheme being used, i.e., row-major ordering or column-major ordering. C language defined storage is specified by **order** = Nag_RowMajor. See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

Constraint: **order** = Nag_RowMajor or Nag_ColMajor.

2: **fact** – Nag_FactoredFormType *Input*

On entry: specifies whether or not the factorized form of the matrix A is supplied on entry, and if not, whether the matrix A should be equilibrated before it is factorized.

fact = Nag_Factored
: **af** and **ipiv** contain the factorized form of A. If **equed** $\neq$ Nag_NoEquilibration, the matrix A has been equilibrated with scaling factors given by **r** and **c**. **a**, **af** and **ipiv** are not modified.

fact = Nag_NotFactored
: The matrix A will be copied to **af** and factorized.

fact = Nag_EquilibrateAndFactor
: The matrix A will be equilibrated if necessary, then copied to **af** and factorized.

Constraint: **fact** = Nag_Factored, Nag_NotFactored or Nag_EquilibrateAndFactor.

3: **trans** – Nag_TransType *Input*

On entry: specifies the form of the system of equations.

trans = Nag_NoTrans
$AX = B$ (No transpose).

trans = Nag_Trans or Nag_ConjTrans
$A^{\mathrm{T}}X = B$ (Transpose).

Constraint: **trans** = Nag_NoTrans, Nag_Trans or Nag_ConjTrans.

4: **n** – Integer *Input*

On entry: n, the number of linear equations, i.e., the order of the matrix A.

Constraint: $\mathbf{n} \geq 0$.

5: **nrhs** – Integer *Input*

On entry: r, the number of right-hand sides, i.e., the number of columns of the matrix B.

Constraint: $\mathbf{nrhs} \geq 0$.

6: **a**[*dim*] – double *Input/Output*

Note: the dimension, *dim*, of the array **a** must be at least $\max(1, \mathbf{pda} \times \mathbf{n})$.

The (i, j)th element of the matrix A is stored in

$\mathbf{a}[(j-1) \times \mathbf{pda} + i - 1]$ when **order** = Nag_ColMajor;
$\mathbf{a}[(i-1) \times \mathbf{pda} + j - 1]$ when **order** = Nag_RowMajor.

On entry: the n by n matrix A.

If **fact** = Nag_Factored and **equed** $\neq$ Nag_NoEquilibration, **a** must have been equilibrated by the scaling factors in **r** and/or **c**.

On exit: if **fact** = Nag_Factored or Nag_NotFactored, or if **fact** = Nag_EquilibrateAndFactor and **equed** = Nag_NoEquilibration, **a** is not modified.

If **fact** = Nag_EquilibrateAndFactor or **equed** $\neq$ Nag_NoEquilibration, A is scaled as follows:

if **equed** = Nag_RowEquilibration, $A = D_R A$;

if **equed** = Nag_ColumnEquilibration, $A = AD_C$;

if **equed** = Nag_RowAndColumnEquilibration, $A = D_R A D_C$.

7: **pda** – Integer *Input*

On entry: the stride separating row or column elements (depending on the value of **order**) in the array **a**.

Constraint: $\mathbf{pda} \geq \max(1, \mathbf{n})$.

8: **af**[*dim*] – double *Input/Output*

Note: the dimension, *dim*, of the array **af** must be at least $\max(1, \mathbf{pdaf} \times \mathbf{n})$.

The (i, j)th element of the matrix is stored in

$\mathbf{af}[(j-1) \times \mathbf{pdaf} + i - 1]$ when **order** = Nag_ColMajor;
$\mathbf{af}[(i-1) \times \mathbf{pdaf} + j - 1]$ when **order** = Nag_RowMajor.

On entry: if **fact** = Nag_Factored, **af** contains the factors L and U from the factorization $A = PLU$ as computed by nag_dgetrf (f07adc). If **equed** $\neq$ Nag_NoEquilibration, **af** is the factorized form of the equilibrated matrix A.

If **fact** = Nag_NotFactored or Nag_EquilibrateAndFactor, **af** need not be set.

On exit: if **fact** $=$ Nag_NotFactored, **af** returns the factors L and U from the factorization $A = PLU$ of the original matrix A.

If **fact** $=$ Nag_EquilibrateAndFactor, **af** returns the factors L and U from the factorization $A = PLU$ of the equilibrated matrix A (see the description of **a** for the form of the equilibrated matrix).

If **fact** $=$ Nag_Factored, **af** is unchanged from entry.

9: **pdaf** – Integer *Input*

On entry: the stride separating row or column elements (depending on the value of **order**) in the array **af**.

Constraint: **pdaf** $\geq \max(1, \mathbf{n})$.

10: **ipiv**$[dim]$ – Integer *Input/Output*

Note: the dimension, *dim*, of the array **ipiv** must be at least $\max(1, \mathbf{n})$.

On entry: if **fact** $=$ Nag_Factored, **ipiv** contains the pivot indices from the factorization $A = PLU$ as computed by nag_dgetrf (f07adc); at the ith step row i of the matrix was interchanged with row **ipiv**$[i-1]$. **ipiv**$[i-1] = i$ indicates a row interchange was not required.

If **fact** $=$ Nag_NotFactored or Nag_EquilibrateAndFactor, **ipiv** need not be set.

On exit: if **fact** $=$ Nag_NotFactored, **ipiv** contains the pivot indices from the factorization $A = PLU$ of the original matrix A.

If **fact** $=$ Nag_EquilibrateAndFactor, **ipiv** contains the pivot indices from the factorization $A = PLU$ of the equilibrated matrix A.

If **fact** $=$ Nag_Factored, **ipiv** is unchanged from entry.

11: **equed** – Nag_EquilibrationType * *Input/Output*

On entry: if **fact** $=$ Nag_NotFactored or Nag_EquilibrateAndFactor, **equed** need not be set.

If **fact** $=$ Nag_Factored, **equed** must specify the form of the equilibration that was performed as follows:

if **equed** $=$ Nag_NoEquilibration, no equilibration;

if **equed** $=$ Nag_RowEquilibration, row equilibration, i.e., A has been premultiplied by D_R;

if **equed** $=$ Nag_ColumnEquilibration, column equilibration, i.e., A has been postmultiplied by D_C;

if **equed** $=$ Nag_RowAndColumnEquilibration, both row and column equilibration, i.e., A has been replaced by $D_R A D_C$.

On exit: if **fact** $=$ Nag_Factored, **equed** is unchanged from entry.

Otherwise, if no constraints are violated, **equed** specifies the form of equilibration that was performed as specified above.

Constraint: if **fact** $=$ Nag_Factored, **equed** $=$ Nag_NoEquilibration, Nag_RowEquilibration, Nag_ColumnEquilibration or Nag_RowAndColumnEquilibration.

12: **r**$[dim]$ – double *Input/Output*

Note: the dimension, *dim*, of the array **r** must be at least $\max(1, \mathbf{n})$.

On entry: if **fact** $=$ Nag_NotFactored or Nag_EquilibrateAndFactor, **r** need not be set.

If **fact** $=$ Nag_Factored and **equed** $=$ Nag_RowEquilibration or Nag_RowAndColumnEquilibration, **r** must contain the row scale factors for A, D_R; each element of **r** must be positive.

On exit: if **fact** $=$ Nag_Factored, **r** is unchanged from entry.

Otherwise, if no constraints are violated and **equed** = Nag_RowEquilibration or Nag_RowAndColumnEquilibration, **r** contains the row scale factors for A, D_R, such that A is multiplied on the left by D_R; each element of **r** is positive.

13: **c**[*dim*] – double *Input/Output*

Note: the dimension, *dim*, of the array **c** must be at least $\max(1, \mathbf{n})$.

On entry: if **fact** = Nag_NotFactored or Nag_EquilibrateAndFactor, **c** need not be set.

If **fact** = Nag_Factored or **equed** = Nag_ColumnEquilibration or Nag_RowAndColumnEquilibration, **c** must contain the column scale factors for A, D_C; each element of **c** must be positive.

On exit: if **fact** = Nag_Factored, **c** is unchanged from entry.

Otherwise, if no constraints are violated and **equed** = Nag_ColumnEquilibration or Nag_RowAndColumnEquilibration, **c** contains the row scale factors for A, D_C; each element of **c** is positive.

14: **b**[*dim*] – double *Input/Output*

Note: the dimension, *dim*, of the array **b** must be at least

$\max(1, \mathbf{pdb} \times \mathbf{nrhs})$ when **order** = Nag_ColMajor;
$\max(1, \mathbf{n} \times \mathbf{pdb})$ when **order** = Nag_RowMajor.

The (i, j)th element of the matrix B is stored in

$\mathbf{b}[(j-1) \times \mathbf{pdb} + i - 1]$ when **order** = Nag_ColMajor;
$\mathbf{b}[(i-1) \times \mathbf{pdb} + j - 1]$ when **order** = Nag_RowMajor.

On entry: the n by r right-hand side matrix B.

On exit: if **equed** = Nag_NoEquilibration, **b** is not modified.

If **trans** = Nag_NoTrans and **equed** = Nag_RowEquilibration or Nag_RowAndColumnEquilibration, **b** is overwritten by D_RB.

If **trans** = Nag_Trans or Nag_ConjTrans and **equed** = Nag_ColumnEquilibration or Nag_RowAndColumnEquilibration, **b** is overwritten by D_CB.

15: **pdb** – Integer *Input*

On entry: the stride separating row or column elements (depending on the value of **order**) in the array **b**.

Constraints:

if **order** = Nag_ColMajor, $\mathbf{pdb} \geq \max(1, \mathbf{n})$;
if **order** = Nag_RowMajor, $\mathbf{pdb} \geq \max(1, \mathbf{nrhs})$.

16: **x**[*dim*] – double *Output*

Note: the dimension, *dim*, of the array **x** must be at least

$\max(1, \mathbf{pdx} \times \mathbf{nrhs})$ when **order** = Nag_ColMajor;
$\max(1, \mathbf{n} \times \mathbf{pdx})$ when **order** = Nag_RowMajor.

The (i, j)th element of the matrix X is stored in

$\mathbf{x}[(j-1) \times \mathbf{pdx} + i - 1]$ when **order** = Nag_ColMajor;
$\mathbf{x}[(i-1) \times \mathbf{pdx} + j - 1]$ when **order** = Nag_RowMajor.

On exit: if **fail.code** = NE_NOERROR or NE_SINGULAR_WP, the n by r solution matrix X to the original system of equations. Note that the arrays A and B are modified on exit if **equed** ≠ Nag_NoEquilibration, and the solution to the equilibrated system is $D_C^{-1}X$ if **trans** = Nag_NoTrans and **equed** = Nag_ColumnEquilibration or

Nag_RowAndColumnEquilibration, or $D_R^{-1}X$ if **trans** = Nag_Trans or Nag_ConjTrans and **equed** = Nag_RowEquilibration or Nag_RowAndColumnEquilibration.

17: **pdx** – Integer *Input*

On entry: the stride separating row or column elements (depending on the value of **order**) in the array **x**.

Constraints:

if **order** = Nag_ColMajor, $\mathbf{pdx} \geq \max(1, \mathbf{n})$;
if **order** = Nag_RowMajor, $\mathbf{pdx} \geq \max(1, \mathbf{nrhs})$.

18: **rcond** – double * *Output*

On exit: if no constraints are violated, an estimate of the reciprocal condition number of the matrix A (after equilibration if that is performed), computed as $\mathbf{rcond} = 1.0/\left(\|A\|_1\|A^{-1}\|_1\right)$.

19: **ferr**[**nrhs**] – double *Output*

On exit: if **fail.code** = NE_NOERROR or NE_SINGULAR_WP, an estimate of the forward error bound for each computed solution vector, such that $\|\hat{x}_j - x_j\|_\infty / \|x_j\|_\infty \leq \mathbf{ferr}[j-1]$ where $\hat{x}_j$ is the jth column of the computed solution returned in the array **x** and x_j is the corresponding column of the exact solution X. The estimate is as reliable as the estimate for **rcond**, and is almost always a slight overestimate of the true error.

20: **berr**[**nrhs**] – double *Output*

On exit: if **fail.code** = NE_NOERROR or NE_SINGULAR_WP, an estimate of the component-wise relative backward error of each computed solution vector $\hat{x}_j$ (i.e., the smallest relative change in any element of A or B that makes $\hat{x}_j$ an exact solution).

21: **recip_growth_factor** – double * *Output*

On exit: if **fail.code** = NE_NOERROR, the reciprocal pivot growth factor $\|A\|/\|U\|$, where $\|.\|$ denotes the maximum absolute element norm. If **recip_growth_factor** $\ll 1$, the stability of the LU factorization of (equilibrated) A could be poor. This also means that the solution **x**, condition estimate **rcond**, and forward error bound **ferr** could be unreliable. If the factorization fails with **fail.code** = NE_SINGULAR, then **recip_growth_factor** contains the reciprocal pivot growth factor for the leading **fail.errnum** columns of A.

22: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_INT

On entry, **n** = ⟨*value*⟩.
Constraint: $\mathbf{n} \geq 0$.

On entry, **nrhs** $= \langle value \rangle$.
Constraint: **nrhs** ≥ 0.

On entry, **pda** $= \langle value \rangle$.
Constraint: **pda** > 0.

On entry, **pdaf** $= \langle value \rangle$.
Constraint: **pdaf** > 0.

On entry, **pdb** $= \langle value \rangle$.
Constraint: **pdb** > 0.

On entry, **pdx** $= \langle value \rangle$.
Constraint: **pdx** > 0.

NE_INT_2

On entry, **pda** $= \langle value \rangle$ and **n** $= \langle value \rangle$.
Constraint: **pda** $\geq \max(1, \mathbf{n})$.

On entry, **pdaf** $= \langle value \rangle$ and **n** $= \langle value \rangle$.
Constraint: **pdaf** $\geq \max(1, \mathbf{n})$.

On entry, **pdb** $= \langle value \rangle$ and **n** $= \langle value \rangle$.
Constraint: **pdb** $\geq \max(1, \mathbf{n})$.

On entry, **pdb** $= \langle value \rangle$ and **nrhs** $= \langle value \rangle$.
Constraint: **pdb** $\geq \max(1, \mathbf{nrhs})$.

On entry, **pdx** $= \langle value \rangle$ and **n** $= \langle value \rangle$.
Constraint: **pdx** $\geq \max(1, \mathbf{n})$.

On entry, **pdx** $= \langle value \rangle$ and **nrhs** $= \langle value \rangle$.
Constraint: **pdx** $\geq \max(1, \mathbf{nrhs})$.

NE_INTERNAL_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.

An unexpected error has been triggered by this function. Please contact NAG.
See Section 3.6.6 in the Essential Introduction for further information.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in the Essential Introduction for further information.

NE_SINGULAR

Element $\langle value \rangle$ of the diagonal is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution and error bounds could not be computed. **rcond** $= 0.0$ is returned.

NE_SINGULAR_WP

U is nonsingular, but **rcond** is less than ***machine precision***, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of **rcond** would suggest.

7 Accuracy

For each right-hand side vector b, the computed solution $\hat{x}$ is the exact solution of a perturbed system of equations $(A+E)\hat{x}=b$, where

$$|E| \leq c(n)\epsilon P|L||U|,$$

$c(n)$ is a modest linear function of n, and ϵ is the ***machine precision***. See Section 9.3 of Higham (2002) for further details.

If x is the true solution, then the computed solution $\hat{x}$ satisfies a forward error bound of the form

$$\frac{\|x-\hat{x}\|_\infty}{\|\hat{x}\|_\infty} \leq w_c \operatorname{cond}(A,\hat{x},b)$$

where $\operatorname{cond}(A,\hat{x},b) = \left\| |A^{-1}|(|A||\hat{x}|+|b|)\right\|_\infty / \|\hat{x}\|_\infty \leq \operatorname{cond}(A) = \left\| |A^{-1}||A|\right\|_\infty \leq \kappa_\infty(A)$. If $\hat{x}$ is the jth column of X, then w_c is returned in **berr**$[j-1]$ and a bound on $\|x-\hat{x}\|_\infty / \|\hat{x}\|_\infty$ is returned in **ferr**$[j-1]$. See Section 4.4 of Anderson *et al.* (1999) for further details.

8 Parallelism and Performance

nag_dgesvx (f07abc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_dgesvx (f07abc) makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the X06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

The factorization of A requires approximately $\frac{2}{3}n^3$ floating-point operations.

Estimating the forward error involves solving a number of systems of linear equations of the form $Ax=b$ or $A^{\mathrm{T}}x=b$; the number is usually 4 or 5 and never more than 11. Each solution involves approximately $2n^2$ operations.

In practice the condition number estimator is very reliable, but it can underestimate the true condition number; see Section 15.3 of Higham (2002) for further details.

The complex analogue of this function is nag_zgesvx (f07apc).

10 Example

This example solves the equations

$$AX=B,$$

where A is the general matrix

$$A = \begin{pmatrix} 1.80 & 2.88 & 2.05 & -0.89 \\ 525.00 & -295.00 & -95.00 & -380.00 \\ 1.58 & -2.69 & -2.90 & -1.04 \\ -1.11 & -0.66 & -0.59 & -0.80 \end{pmatrix}$$

and

$$B = \begin{pmatrix} 9.52 & 18.47 \\ 2435.00 & 225.00 \\ 0.77 & -13.28 \\ -6.22 & -6.21 \end{pmatrix}.$$

Error estimates for the solutions, information on scaling, an estimate of the reciprocal of the condition number of the scaled matrix A and an estimate of the reciprocal of the pivot growth factor for the factorization of A are also output.

10.1 Program Text

```
/* nag_dgesvx (f07abc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 */

#include <stdio.h>
#include <nag.h>
#include <nagx04.h>
#include <nag_stdlib.h>
#include <nagf07.h>

int main(void)
{

  /* Scalars */
  double                growth_factor, rcond;
  Integer               exit_status = 0, i, j, n, nrhs, pda, pdaf, pdb, pdx;

  /* Arrays */
  double                *a = 0, *af = 0, *b = 0, *berr = 0, *c = 0, *ferr = 0;
  double                *r = 0, *x = 0;
  Integer               *ipiv = 0;

  /* Nag Types */
  NagError              fail;
  Nag_OrderType         order;
  Nag_EquilibrationType equed;

#ifdef NAG_COLUMN_MAJOR
#define A(I, J) a[(J-1)*pda + I - 1]
#define B(I, J) b[(J-1)*pdb + I - 1]
  order = Nag_ColMajor;
#else
#define A(I, J) a[(I-1)*pda + J - 1]
#define B(I, J) b[(I-1)*pdb + J - 1]
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);

  printf("nag_dgesvx (f07abc) Example Program Results\n\n");

  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%"NAG_IFMT"%*[^\n]", &n, &nrhs);
#else
  scanf("%"NAG_IFMT"%"NAG_IFMT"%*[^\n]", &n, &nrhs);
#endif

  if (n < 0 || nrhs < 0)
    {
      printf("Invalid n or nrhs\n");
      exit_status = 1;
      return exit_status;
    }
  pda = n;
  pdaf = n;
#ifdef NAG_COLUMN_MAJOR
  pdb = n;
  pdx = n;
#else
  pdb = nrhs;
  pdx = nrhs;
#endif

  /* Allocate memory */
  if (
    !(a  = NAG_ALLOC(n * n, double)) ||
    !(af = NAG_ALLOC(n * n, double)) ||
    !(b  = NAG_ALLOC(n * n, double)) ||
    !(berr = NAG_ALLOC(n, double)) ||
    !(c  = NAG_ALLOC(n, double)) ||
    !(ferr = NAG_ALLOC(n, double)) ||
    !(r  = NAG_ALLOC(n, double)) ||
    !(x  = NAG_ALLOC(n*n, double)) ||
    !(ipiv = NAG_ALLOC(n, Integer)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Read A and B from data file */

  for (i = 1; i <= n; ++i)
#ifdef _WIN32
    for (j = 1; j <= n; ++j) scanf_s("%lf", &A(i, j));
#else
    for (j = 1; j <= n; ++j) scanf("%lf", &A(i, j));
#endif
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  for (i = 1; i <= n; ++i)
#ifdef _WIN32
    for (j = 1; j <= nrhs; ++j) scanf_s("%lf", &B(i, j));
#else
    for (j = 1; j <= nrhs; ++j) scanf("%lf", &B(i, j));
#endif
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  /* Solve the equations AX = B for X using
   * nag_dgesvx (f07abc)
   */
  nag_dgesvx(order, Nag_EquilibrateAndFactor, Nag_NoTrans, n, nrhs, a, pda, af,
             pdaf, ipiv, &equed, r, c, b, pdb, x, pdx, &rcond, ferr, berr,
             &growth_factor, &fail);

  if (fail.code != NE_NOERROR && fail.code != NE_SINGULAR)
    {
      printf("Error from nag_dgesvx (f07abc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Print solution using
   * nag_gen_real_mat_print (x04cac)
   */
  fflush(stdout);
  nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, nrhs, x,
                         pdx, "Solution(s)", 0, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Print error bounds, condition number, the form of equilibration
   * and the pivot growth factor
   */
  printf("\nBackward errors (machine-dependent)\n");
  for (j = 1; j <= nrhs; ++j)
    printf("%11.1e%s", berr[j - 1], j%7 == 0 || j == nrhs?"\n":" ");
  printf("\n\nEstimated forward error bounds (machine-dependent)\n");

  for (j = 1; j <= nrhs; ++j)
    printf("%11.1e%s", ferr[j - 1], j%7 == 0 || j == nrhs?"\n":" ");
  printf("\n");

  if (equed == Nag_NoEquilibration)
    printf("A has not been equilibrated\n");
  else if (equed == Nag_RowEquilibration)
    printf("A has been row scaled as diag(R)*A\n");
  else if (equed == Nag_ColumnEquilibration)
    printf("A has been column scaled as A*diag(C)\n");
  else if (equed == Nag_RowAndColumnEquilibration)
    printf("A has been row and column scaled as diag(R)*A*diag(C)\n");

  printf("\nReciprocal condition number estimate of scaled matrix\n");
  printf("%11.1e\n\n", rcond);

  printf("Estimate of reciprocal pivot growth factor\n");
  printf("%11.1e\n", growth_factor);

  if (fail.code == NE_SINGULAR)
    {
      printf("Error from nag_dgesvx (f07abc).\n%s\n", fail.message);
      exit_status = 1;
    }

 END:
  NAG_FREE(a);
  NAG_FREE(af);
  NAG_FREE(b);
  NAG_FREE(berr);
  NAG_FREE(c);
  NAG_FREE(ferr);
  NAG_FREE(r);
  NAG_FREE(x);
  NAG_FREE(ipiv);

  return exit_status;
}
#undef B
#undef A
```

10.2 Program Data

```
nag_dgesvx (f07abc) Example Program Data

    4        2                     :Values of n and nrhs

    1.80     2.88    2.05    -0.89
  525.00  -295.00  -95.00  -380.00
    1.58    -2.69   -2.90    -1.04
   -1.11    -0.66   -0.59     0.80 :End of matrix A
```

```
    9.52    18.47
 2435.00   225.00
    0.77   -13.28
   -6.22    -6.21                   :End of matrix B
```

10.3 Program Results

```
nag_dgesvx (f07abc) Example Program Results

 Solution(s)
            1          2
 1      1.0000     3.0000
 2     -1.0000     2.0000
 3      3.0000     4.0000
 4     -5.0000     1.0000

Backward errors (machine-dependent)
    6.8e-17     9.1e-17


Estimated forward error bounds (machine-dependent)
    2.4e-14     3.6e-14

A has been row scaled as diag(R)*A

Reciprocal condition number estimate of scaled matrix
    1.8e-02

Estimate of reciprocal pivot growth factor
    7.4e-01
```